Representing Distributed Groups with $_{dg}$FOAF

Felix Schwagereit, Ansgar Scherp, Steffen Staab

WeST Institute, University of Koblenz-Landau, Germany
{schwagereit,scherp,staab}@uni-koblenz.de

Abstract. Managing one's memberships in different online communities increasingly becomes a cumbersome task. This is due to the increasing number of communities in which users participate and in which they share information with different groups of people like colleagues, sports clubs, groups with specific interests, family, friends, and others. These groups use different platforms to perform their tasks such as collaborative creation of documents, sharing of documents and media, conducting polls, and others. Thus, the groups are scattered and distributed over multiple community platforms that each require a distinct user account and management of the group. In this paper, we present $_{dg}$FOAF, an approach for distributed group management based on the well known Friend-of-a-Friend (FOAF) vocabulary. Our $_{dg}$FOAF approach is independent of the concrete community platforms we find today and needs no central server. It allows for defining communities across multiple systems and alleviates the community administration task. Applications of $_{dg}$FOAF range from access restriction to trust support based on community membership.

1 Introduction

An increasing number of Web 2.0 platforms providing social networking and community building functionality are available. Users participate in multiple of such platforms and share information with different groups of people of their social network such as the professional network, sports club, colleagues, special interest groups, friends, family, and others. This is due to network effects, i.e., users need to join platforms other users are already members in. In addition, the platforms provide different functionality and have different scope such as building a professional network, sharing media, or conducting polls. In order to conduct the tasks within a specific group of the social network, multiple community platforms need to be used. Thus, the users require accounts in several community platforms and the groups the users are members in are scattered over different of such platforms.

Communities are considered in this work as set of people that share a common purpose [7]. This is the case for a long-lasting professional organization like the W3C, which has the goal of creating web standards, as well as for an ad-hoc created group of private persons organizing a party. The members of a community may be recognized by their interests, behavior, or contacts. Thus

they form an implicit community since their membership is not explicitly defined. If membership in a community is managed intentionally, an explicitly defined community is formed. This paper focuses on communities that explicitly specify their members. In the following, such communities are called *groups*. A person may be involved in many different groups such as professional networks, sports clubs, and groups with specific interests. If these groups are distributed over different systems, the management of group memberships becomes a tedious task. Therefore, a decentralized approach for managing group memberships is an appropriate way to tackle these problems.

The $_{dg}$FOAF approach assumes the following setting: People can describe themselves with the Friend-of-a-Friend (FOAF) vocabulary [3]. We assume that personal descriptions as well as other statements are located at Linked Open Data (LOD) stores [4] owned by the particular person. So once the location of the self description of a particular person is known and confirmed (e.g., using signed graphs [10]) it can be assumed that all the data available at this location is stated by this person. Based on this self description, several approaches like OpenID [8] or FOAF+SSL [9] exist to authenticate persons. Different Web-based services can use these authentication mechanisms for providing Web 2.0 functionality as indicated in [13]. The contribution of this paper is $_{dg}$FOAF, a policy based mechanism for defining distributed groups and for determining group membership. $_{dg}$FOAF provides functionality sufficient to support typical Web 2.0 scenarios of group administration as well as other Internet-based collaboration tools.

2 Motivating Scenario

In a typical group management scenario, several people need to organize their actions to pursue a common goal, which is in our case the organization of an event. The scenario is motivated from the "Consumer Social Group" use case of the WeKnowIt[1] project: The 1999 class of the Westpark High School graduated 10 years ago. Alice thinks it is time for a class reunion. She asks her former classmate Bob to help her organizing such an event. (i) To prepare the class reunion, Alice and Bob set up a group named WestparkHigh99, which is going to be populated by all known members of their old class. (ii) Together Alice and Bob form the organizing committee for the class reunion, the group WestparkHigh99Committee. (iii) During their ongoing preparations Alice and Bob persuade Carol to join the organizing committee and further classmates are found. (iv) Carol adds Ronald and Bob adds Simon to the group. Over time, the group grows as more and more classmates are discovered. Later Alice realizes that Ronald is not the person they had assumed. (v) Since Carol is temporarily not available and cannot correct her mistake, Alice removes Ronald from the group. (vi) Alice, Bob, and Carol set up online services like a photo sharing system to collect old photos and a scheduler to find a date for their reunion. These services exist on different specialized platforms like Flickr (http://flickr.com) and

[1] http:www.weknowit.eu

Doodle (http://doodle.com). Access to these services is granted only to members of the WestparkHigh99 group.

3 Problem Areas of the Scenario

We investigate how the scenario can be implemented with currently available means. We focus on the required infrastructure and the actions that are carried out by the involved people. These actions are creation of groups and modification of group membership. Also the group membership of people is determined in order to decide whether they are allowed to access specific services.

3.1 Centralized Solution

In a centralized solution, Alice would start to search for online services she needs for organizing the reunion. Probably she wants to use the services "Photo Sharing" and "Meeting Scheduler" along with other services. Then she would create a group WestparkHigh99 at every chosen service with herself as administrator. Later she adds Bob and Carol as administrators for her group at each service and the administrators add the former classmates to each of the services. So each service can determine through its own group structures whether a specific person is allowed access and using it.

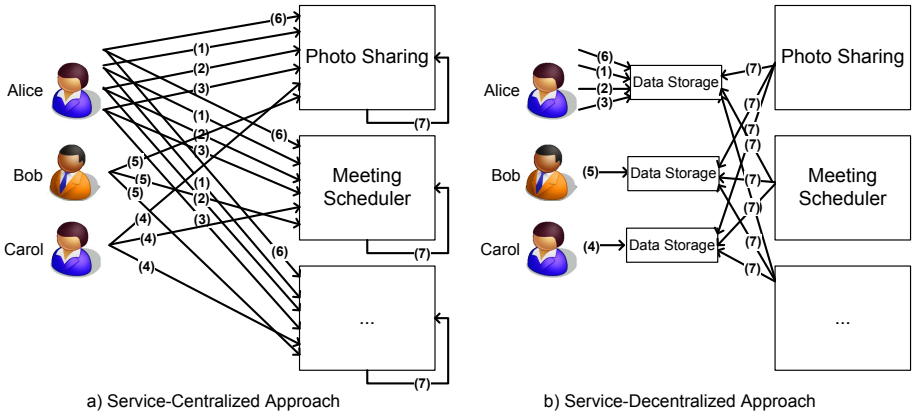

Fig. 1. Service-Centralized Approach vs. Service-Decentralized Approach

The actions are depicted in Figure 1 a) and numbered in the order of execution: (1) create group WestparkHigh99, (2) add Bob as admin for WestparkHigh99, (3) add Carol as admin for WestparkHigh99, (4) add Ronald as member of WestparkHigh99, (5) add Simon as member of WestparkHigh99, (6) remove Ronald from the group WestparkHigh99, (7) determine membership in

WestparkHigh99. As shown in Figure 1 a), all actions need to be carried out for each service platform separately. Also the users have no common control over changes of the end user policies at the service platforms. Once they created all group structures they are locked in that platform, because otherwise they would have to recreate all the structures again on another platform. Thus, the centralized solution where all actions are centered around individual service platforms has several disadvantages:

- High effort to maintain different identical group structures
- Error-prone because of redundancy
- Lock-in effect

3.2 Decentralized Solution

To overcome the disadvantages of the centralized solution, we separate the group management from the services that are used by the group to carry out their tasks. In such a decentralized solution, the actions for the group management need to be carried out only once as depicted in Figure 1 b). The numbers match the actions described in the previous section.

In the decentralized solution a data storage is introduced, which is open for online read access but owned by one person. Every person can conduct group management tasks by modifying her own data storage. This requires fewer actions from the people involved. On the other side, the platforms are required to conduct more actions, since the have to "construct" the current status of the group from different data storages. New questions arise, if group management is decentralized in the way described above:

- Can multiple administrators act on behalf of one, single group? Although it is easy for every administrator to define her own group in her own data storage, this is not sufficient for multiple administrators, who cannot modify each others' data.
- Where is the group's policy located? The policy describes which actions can be conducted by whom. But since different administrators act on the same group they all need to express that they do so without any central entity.
- How can an administrator undo the action of another administrator? Since no person can modify the actions stated in data storages owned by other people, deletion is not applicable.
- How can the service platforms determine the membership of one person? This task is not trivial since data needs to be combined that comes from different data storages and might therefore be contradicting.

In a naive alternative approach, all group management tasks could be conducted on one single data storage. This would solve the problem of multiple data storages, but the problem of managing this one data storage as well as the problem of multiple platforms used by different groups would remain. So this naive approach would not completely address our problem statement. The challenges resulting from these questions are tackled with the integrated $_{dg}$FOAF approach.

4 Requirements

In this section, we collect requirements from the scenario while considering the open questions mentioned in the previous section. These requirements will form a baseline for the architecture and functionality of $_{dg}$FOAF. For each requirement, we also explicitly refer to the scenario in Section 2 by referring to the (<number>) of the relevant part.

1. *Group Administrators:* For a group, one administrator or a set of administrators can be defined. The administrators are allowed to modify memberships of people. In the scenario, Alice and Bob have initially created the group WestparkHigh99 with themselves as administrators (i).
2. *Granting of Membership:* Administrators can grant group membership to persons or members of other groups. In the scenario, Bob is an administrator. So he adds Simon to WestparkHigh99 (iv).
3. *Banning of Members:* Administrators can ban persons from groups. In the scenario, Alice bans the wrong Ronald from the group (v).
4. *Management of Group Administration:* New administrators can be appointed or existing administrators can resign. Since Carol becomes member of the WestparkHigh99Committee, the administrators of the WestparkHigh99 group have changed (iii).
5. *No Super Administrators:* For administration of a group, super-administrators, i.e., a person who administrates administrators, are not mandatory. In the scenario, the group WestparkHigh99Committee is not administrated by persons other than members of the group itself (iii).
6. *Determination of Group Memberships:* A mechanism needs to be provided that determines the group membership of a person based on the group definition and the actions of the group administrators. In the scenario, the services set up for WestparkHigh99determine the group membership of every person who wants to use them (vi).
7. *Referencing of Groups:* Especially in a distributed environment, a group needs an unambiguous identifier that can be referenced by services based on group membership. The group and group memberships should be stated in such a way that, if one definition is deleted, the remaining parts of the group continue to exist. In the scenario, Bob and Carol can still add group members, although Alice's part is not available (vi).

5 The $_{dg}$FOAF Approach

We introduce the foundational design of $_{dg}$FOAF by distinguishing stable and changing group characteristics, which lead to the concepts of group policy and membership definition. This design is a decentralized solution described in Section 3.2.

Groups are immaterial entities. They only come into existence by making statements about them or in other words by describing them. When all statements about a group are combined, they reveal all characteristics of a group.

Certainly the most interesting characteristics of a group for practical use are its name and its members. We distinguish between two general types of group characteristics: the characteristics that are stable from creation to deletion of the group and the characteristics that can change.

In $_{dg}$FOAF all stable characteristics of a group are called group policy, which has to be stated explicitly. So, by definition, a group policy cannot change for one specific group, although it is possible to recreate a group that has a slightly different group policy. The group policy contains at least one name. Although such a name cannot be guaranteed to be unique, it can be chosen such that the group cannot be confused with another similar group. Other characteristics, which are not to be changed through the life of a group, are the rules specifying how membership can be achieved. In $_{dg}$FOAF these rules are conditions that a person has to satisfy in order to affect group membership of herself or others.

Membership of persons in a group is the only characteristic of groups covered in $_{dg}$FOAF that can change through the life of a group. By stating membership definitions a person can add or remove members from the group. In contrast to other linked data, the statements about $_{dg}$FOAF groups can be evaluated by taking into account whether the person who made the statements meets the explicit conditions in the group's policy. Therefore different and possibly conflicting membership definitions can be checked, so that for each combination of membership statements only one single interpretation can be given. Therefore the group policy acts as the basis for deciding whether a membership definition by a specific person is going to affect the group characteristics.

6 Concepts and Schema of $_{dg}$FOAF

Based on the requirements discussed in Section 4, we introduce all concepts of $_{dg}$FOAF that are needed to describe groups according to the design approach of Section 5. Further, we introduce a Resource Description Framework (RDF)[2] schema for serialization of these concepts and illustrate its usage with an example from the scenario. This schema is depicted in Figure 2. It contains all concepts that are needed to express $_{dg}$FOAF groups. Besides the namespace `foaf` for elements already defined in the standard FOAF vocabulary, the namespace `dgfoaf` is introduced to define elements added for $_{dg}$FOAF. From the scenario given in Section 2, we have extracted a fragment of Alice's $_{dg}$FOAF profile. It is presented in Figure 3. In the following, we introduce the concepts of $_{dg}$FOAF based on this profile.

Persons are atomic entities in $_{dg}$FOAF, who act as owners of data storages and therefore of the groups stored there. In the scenario, Alice is defined as a person, who is owner of the data storage at the location http://alice.com/me.ttl. In Alice's $_{dg}$FOAF profile shown in Figure 3 this is represented at lines 2-4.

Groups are entities with two explicit and stable properties. The first property is the group owner, the person who owns the group's location, i.e., the group's

[2] http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

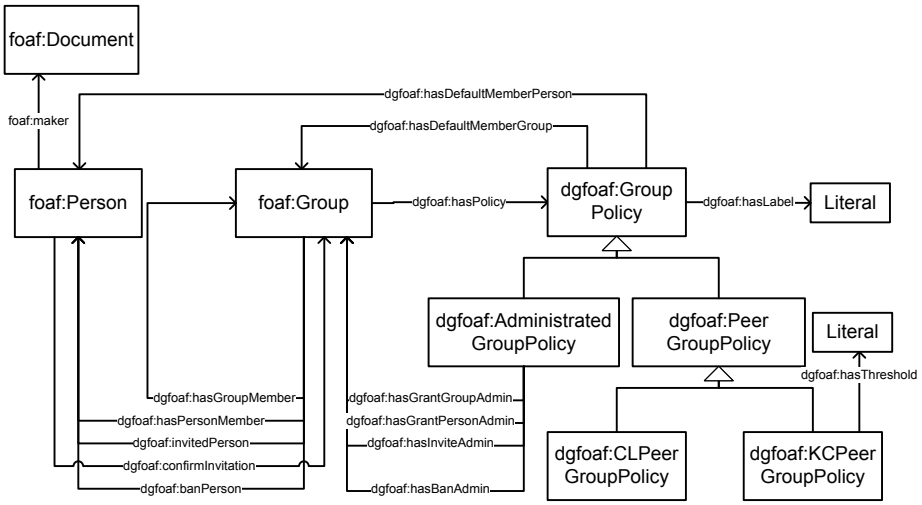

Fig. 2. $_{dg}$FOAF Schema

URI. The second stable property of a group is its group policy. So groups are structures, consisting of one person (the group owner) and one policy. In the scenario, Alice defines two groups WestparkHigh99 and WestparkHigh99Committee. These two groups can be found in Alice's $_{dg}$FOAF profile at lines 6 and 14. The corresponding policies for the two groups are referenced at lines 7 and 15.

Group Policies have various properties. One of these properties is their label, which is an arbitrary string. Different labels allow giving groups with identical policies different properties in order to distinguish between them. In Alice's $_{dg}$FOAF profile, two labels for two groups are defined at lines 10 and 19. Group policies can also define specific persons (defaultMemberPerson) or the members of specific groups (defaultMemberGroup) to be members of the group, without the need for membership grants. The two default member properties are available for every policy type.

Two types of group policies can be distinguished: policies for administrated groups and policies for peer groups. For *policies for administrated groups*, restrictions can be made about what administrative actions are allowed to be made by which persons. These persons need to be member of other groups, called admin groups. Admin groups can exist for granting membership to persons (grantPersonAdmin), inviting persons (inviteAdmin), granting membership to groups (grantGroupAdmin), and banning persons from the group (banAdmin). In the scenario, Alice's group WestparkHigh99 has an administrative group policy defined at lines 9-12. Lines 11 and 12 define the group WestparkHigh99Committee as admin group for granting membership and banning persons. *Policies for peer groups* can be divided into two types: While a CliquePeerGroupPolicy (`dgfoaf:CLPeerGroupPolicy`) defines a group in which members need to be connected to every other member, the K-Core-PeerGroupPolicy

```
1  @prefix : <http://alice.com/me.ttl#> .
2  :me a foaf:Person ;
3          foaf:name "Alice" .
4  <http://alice.com/me.ttl> foaf:maker :me .
5
6  :WestparkHigh99 a foaf:Group ;
7          dgfoaf:hasPolicy :WestparkHigh99Policy ;
8          dgfoaf:banPerson <http://ronald.com/me.ttl#me> .
9  :WestparkHigh99Policy a dgfoaf:AdministratedGroupPolicy ;
10         dgfoaf:hasLabel "WestParkHigh99Group" ;
11         dgfoaf:hasGrantPersonAdmin :WestparkHigh99Committee ;
12         dgfoaf:hasBanAdmin :WestparkHigh99Committee .
13
14 :WestparkHigh99Committee a foaf:Group ;
15         dgfoaf:hasPolicy :WestparkHigh99ComPolicy ;
16         dgfoaf:hasPersonMember <http://bob.com/me.ttl#me>,
17                 <http://carol.com/me.ttl#me> .
18 :WestparkHigh99ComPolicy a dgfoaf:CLPeerGroupPolicy ;
19         dgfoaf:hasLabel "WestParkHigh99CommitteeGroup" .
```

Fig. 3. Alice's $_{dg}$FOAF Profile in Turtle Syntax

(`dgfoaf:KCPeerGroupPolicy`) defines a group in which all members need to be connected to at least K other members. In the scenario, Alice defines the group WestparkHigh99Committee as having a Clique-PeerGroupPolicy in line 18. Therefore every other member has to grant membership to Alice in order that Alice becomes member of WestparkHigh99Committee and vice versa.

Membership Definitions are assertions, made by the group owner, about the membership of other persons in her group. If the owner is allowed to assert them they result in a group membership change. Since membership definitions can change over time they are not part of the group policy. For groups with a policy for administrated groups, the following membership definitions are available: grant membership to persons (hasPersonMember), invite persons (invitedPerson), grant membership to members of other groups (hasGroupMember) and ban persons from the group (banPerson). Note that an explicit distinction between persons and groups being members of a group has to be made to avoid affects resulting from a type change of the referenced person to group. For example, if Alice grants membership to Bob, he should not be able to grant membership to other persons by simply changing the type of the entity representing him as `foaf:Person` to `foaf:Group` and adding persons to this misused group. In Alice's $_{dg}$FOAF profile she bans Ronald from being a member of WestparkHigh99 in line 8. For groups with a policy for peer groups, only the membership definition hasPersonMember is available for granting membership to other persons. In the scenario, Alice has granted such membership to Bob and Carol for group WestparkHigh99Committee in lines 16 and 17.

Basics:

$Persons := the\ set\ of\ all\ Persons$
$Labels := the\ set\ of\ all\ Strings$
$Groups := Persons \times Policies$

Policies:

$$\begin{aligned}
&GeneralGroupPolicyProperties \\
&\quad := \{defaultMemberPerson\} \times Persons \cup \\
&\quad\quad \{defaultMemberGroup\} \times Groups \cup \\
&\quad\quad Labels \\
&AdministratedGroupPolicyProperties \\
&\quad := \{hasGrantPersonAdmin, hasGrantGroupAdmin, \\
&\quad\quad hasInviteAdmin, hasBanAdmin\} \times Groups \\
&AdministratedGroupPolicies \\
&\quad := \{pr_i\}_{i\in\{1,\ldots,k\}} |\ pr \in AdministratedGroupPoliyProperties \cup \\
&\quad\quad GeneralGroupPolicyProperties, k \in \mathbb{N} \\
&CliquePeerGroupPolicies \\
&\quad := \{pr_i\}_{i\in\{1,\ldots,k\}} \mid pr \in GeneralGroupPolicyProperties, k \in \mathbb{N} \\
&KCoreGroupPolicies \\
&\quad := \{pr_i\}_{i\in\{1,\ldots,k\}} \ \cup \{\{threshold\} \times t\} \\
&\quad\quad \mid pr \in GeneralGroupPolicyProperties, k \in \mathbb{N}, t \in \mathbb{N} \\
&Policies := AdministratedGroupPolicies \cup CliquePeerGroupPolicies \cup \\
&\quad\quad KCorePeerGroupPolicies
\end{aligned}$$

Membership Definitions:

$$\begin{aligned}
&PersonMembershipDefinitions \\
&\quad := \{personMembershipDefinition, banPersonDefinition, \\
&\quad\quad invitePersonDefinition, invitationConfirmationDefintion\} \\
&\quad\quad \times Persons \times Groups \times Persons \\
&GroupMembershipDefinitions \\
&\quad := \{groupMembershipDefinition\} \times Persons \times Groups \times Groups \\
&MembershipDefinitions \\
&\quad := PersonMembershipDefinitions \cup GroupMembershipDefinitions
\end{aligned}$$

Table 1. Policies and Membership Definitions for $_{dg}$FOAF Groups

The elements of $_{dg}$FOAF described above are summarized in Table 1. Basic concepts are *Persons*, *Groups* consisting of one person and one policy, and *Labels* consisting of arbitrary strings. In order to define *Policies* it is required to define their properties. *GeneralGroupPolicyPropterties* can be used in policies of every type. *AdministratedGroupPolicyProperties* are used in *AdministratedGroupPolicies*. The *CliquePeerGroupPolicies* can only have general properties, while the *KCorePeerGroupPolicies* additionally have one *threshold*. All elements of these three policy types finally represent the *Policies* in $_{dg}$FOAF. *Membership Definitions* in $_{dg}$FOAF can be distinguished in *PersonMembershipDefinitions* and *GroupMembershipDefinitions*. *PersonMembershipDefinitions* are a tuple of one policy type (person membership, ban, invitation, confirmation), the person who issued them, i.e., defined it in his FOAF file, the group that is to be addressed, and the person who is the object of this definition. *GroupMembershipDefinitions*

are a tuple of the type *groupMembershipDefinitionType*, the person who issued them, i.e., defined it in his FOAF file, the group A that is to be addressed, and group B whose members is granted membership in group A.

7 Determination of Group Memberships in $_{dg}$FOAF

The semantics of $_{dg}$FOAF is expressed in the methods that are used to determine membership of a given person based on all group policies and the membership definitions. So the semantics can be entirely reduced to the description of the membership function. The membership function has two parts. Part one is the merging of identical group policies. In the second part, the membership of a person in a specific group is determined.

7.1 Merging of Group Policies

In a centralized group management, the definition of a group and its members is stored in a single location. In a distributed group management, this single location cannot be assumed, especially when more than one administrator is responsible for managing the group. Choosing a single location point would then result in an administrator who is owning that location and the other administrator has no or only limited access to it. To overcome these drawbacks, in $_{dg}$FOAF there is no single location where the group definition is stored and no distinction between different "classes" of administrators. In order to manage a group by multiple administrators, they have to agree on the same group policy. Thus, every administrator has to be able to maintain a copy of the group policy in his $_{dg}$FOAF profile. If such a group has exactly the same group policy as the group maintained by another administrator, both groups are considered to be one single group. That results in merging all memberships in the two groups to membership in one group. Therefore in $_{dg}$FOAF there is implemented the function $isIdentical : Group \times Group \mapsto \{true, false\}$ which returns true if both group policies are identical and false otherwise. In order to be identical, the two policies are required to have exactly the same properties. Furthermore, the properties of the two policies are required to have identical values. We distinguish three possible types of values: literals, *Person*s, and *Group*s. Values of literals and persons are required to be identical. To compare values of type *Group* the function *isIdentical* is used recursively. Please note that based on this definition, two groups can be considered identical because of their identical policies although they have different owners.

Based on the function *isIdentical*, another function is defined that assigns to each group a so called general identifier. Two groups that have identical policies are therefore assigned the same general identifier: $generalIdentifier : Group \mapsto GeneralIdentifier$. The general identifier is used in the membership function to abstract from different groups with identical policies, so that they can be processed as one single group. In the following, all definitions of groups and membership are considered to be made about groups referenced by their general identifier.

7.2 Membership Function

The determination of group membership of a person in a group is the core component of $_{dg}$FOAF. The methods for determining membership differ between the two general types of groups, namely administrated groups and peer groups. Nevertheless some properties are identical for both types of groups. In order to preserve the provenance of every membership definition, the definitions are labeled with the person who issued them, i.e., in whose $_{dg}$FOAF profile they are defined. Membership in a group can be defined by a default policy. Thus, it cannot be altered without affecting the merging with other groups, c.f. Section 7.1. If default membership is granted to a person, this person is a member in every configuration of the group. Such persons cannot be banned or otherwise removed. Two types of default membership can be distinguished: *defaultMemberPerson* and *defaultMemberGroup*. A *defaultMemberPerson* can only be a reference to a person node. This person is in every case determined to be a member of the particular group. The *defaultMemberGroup* references another group B. All members of group B are then automatically members of the particular group.

Administrated Groups can have several administrators. Each administrator is allowed issuing specific membership definitions. Only the allowed membership definitions can affect the membership of persons in the group. The group policy defines who is allowed to issue membership definitions. This is done by explicitly declaring a group as an admin group. Members of such a group get the respective administration role. Consequently, the membership of a person in an admin group needs to be verified in order to verify if this person has got the corresponding administration right. We distinguish administration roles for the following types of membership definitions:

- Granting membership to specific persons
- Inviting specific persons to the group
- Granting membership to all members of specific groups
- Banning persons as members of the group

Corresponding to these different roles, four different types of functions are generated that assign a truth value to a tuple of person and group, e.g., $canGrantPersonMembership : Person \times Group \mapsto \{true, false\}$ that is true for all persons who have the role of granting membership to specific persons. Based on these functions, all membership definitions can be interpreted in the following way: A membership definition issued by a person that is allowed to do so is valid, otherwise it will be ignored. The valid membership definitions of a group are the basis for determining membership of a specific person in that group. Three ways of gaining membership in an administrated group exist:

- A valid membership definition exists, which grants membership to a specific person. In addition, there exists no valid membership definition, which bans the person from that group.
- A valid membership definition exists, which invites a person to the group. This person has issued a confirmation of invitation and there exists no valid membership definition, which bans the person from that group.

– A valid membership definition exists, which grants membership to members of another specified group, e.g., group B. Then the members of group B are also members in the group, if there exists no valid membership definition, which bans the person from that group.

Peer Groups are not administrated by members of other specific groups but directly by the members of the peer group itself. Membership in a peer group is based on membership definitions that have been issued by persons. These membership definitions are interpreted as a social network. So methods for constructing cohesive subgroups can be applied. Suitable methods for peer groups are based on the degree of each person, i.e., the number of connections of each person to other persons. There also exist methods that are based on other measures like path length. But since these measures are difficult to control from the local view point of each person, they are less suitable than the degree property used in the following.

As described in Section 6, we distinguish two types of peer groups according to their policy: K-Core-PeerGroupPolicy and Clique-PeerGroupPolicy. The K-Core Policy requires every member to have a mutual connection to at least K other members. The threshold K is a property of any K-Core Policy. So a social network analysis algorithm for detecting k-cores [12] can be applied. The Clique Policy requires that every member has a mutual connection to every other member. The result of the applied algorithms is a set of persons who are in the particular peer group.

8 $_{dg}$FOAF for an Access Control Application

In this section, we show how $_{dg}$FOAF can be embedded in an architecture that integrates authentication and authorization for restricting access to web resources. The authentication is done via a FOAF-based authentication mechanism like FOAF+SSL [9] or with OpenID [8]. The authorization step is based on a query of group membership in a $_{dg}$FOAF group. We illustrate this architecture at the example of a picture sharing service in the scenario of Section 2. The architecture is depicted in Figure 4. In this scenario, Simon wants to access resources of the picture sharing service, which are restricted to members of the group WestparkHigh99. In order to allow Simon to access the resources, authentication and authorization needs to be performed by the picture sharing service.

Authentication Before the system can decide whether an access request to its resources can be allowed, it needs to clarify the identity of the person. For authentication, a FOAF compatible mechanism like FOAF+SSL or OpenID is used (task A). So the person who wants to be authenticated has to prove that he is the owner of a specific FOAF file. In case of FOAF+SSL, this is done by knowledge of a secret key, which is used during HTTPS authentication. With OpenID, this is done by proving his identity to an external service. If the authorization was successful, Simon has proven that the person who wants access is the person described in Simon's FOAF profile.

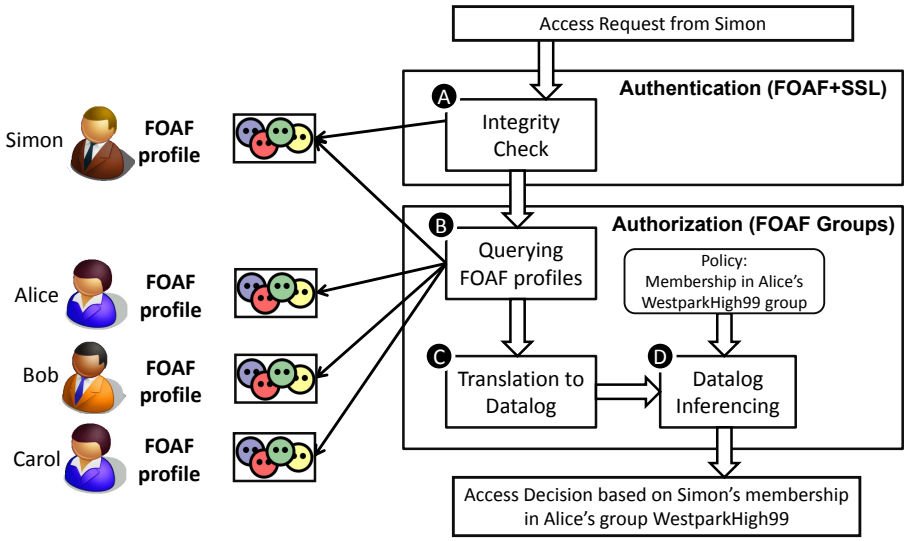

Fig. 4. Elements of Membership Determination

Authorization Knowing his FOAF-based identity, the picture sharing service needs to decide whether Simon is allowed to gain access. In the scenario, access is only granted to members of the group WestparkHigh99. Therefore the system needs to determine Simon's membership in this group. Therefore in task B, all FOAF profiles that contain information about members of the group WestparkHigh99 are queried for their group definitions. In our scenario, we assume that Alice, Bob, Carol, and Simon have created their personal $_{dg}$FOAF profiles with an appropriate editor. These FOAF profiles are queried for membership determination. The task of discovering the needed FOAF profiles in order to determine a particular membership can be non-trivial and complex for other scenarios. Therefore techniques similar to proof-carrying authentication [1] may be applied, which are outside the scope of this paper. Subsequently the collected $_{dg}$FOAF group definitions are translated to a declarative language in task C. We use Datalog as it allows for a declarative representation of $_{dg}$FOAF semantics. The generated Datalog statements constitute a knowledge base used in the next task. Finally in task D, a Datalog inference engine queries the knowledge base for the membership of Simon in the group WestparkHigh99. According to the result of the query Simon is granted access to the photos uploaded to the picture sharing service.

We have implemented the described authorization step as a prototype in Java and defined the processes as Datalog rules. The relevant parts of $_{dg}$FOAF profiles are retrieved via defined SPARQL queries using Sesame[3] and then translated to

[3] http://www.openrdf.org/

Datalog facts. We use the DLV[4] interpreter as Datalog engine. For easy use of $_{dg}$FOAF within existing platforms the service of determining membership of a person could also be provided by a trusted third party. Therefore, the community platform would not need to integrate the membership determination itself.

9 Related Work

FOAF [3] is a well known vocabulary for describing attributes of persons and relationships between persons. A person can create one (or more) FOAF profiles to provide information about herself. Popular properties are `foaf:name`, `foaf:homepage`, and `foaf:mbox`. The `foaf:knows` property is often used to state a relationship to other persons. Groups can be defined in FOAF as instances of the type `foaf:Group`. With `foaf:membershipClass` and `foaf:member`, persons can be assigned to a group. Although it is possible to infer not explicitly given group members by a class definition of members, no formal specification is given for implications of group membership in FOAF. In the WIQA framework [2], provenance of an arbitrary statement is considered as one parameter for accessing trust in this statement. While $_{dg}$FOAF provides a specification for expressing group structures, the WIQA framework is more generic. The OpenSocial application programming interface (API)[5] is an effort led by Google to standardize an interface for profiles that are stored at social network platforms. Applications based on this API can be executed on specific social network platforms using the available data. They do not allow for exchange and modification of group specifications.

Our $_{dg}$FOAF approach provides functionality similar to what could be retrieved out of a combination of existing more general approaches for policy-based trust management frameworks (SD3 [5], RT [6], KAOS [11]). But this specific combination along with its restriction to specific policy alternatives for describing groups is the strength of $_{dg}$FOAF. SD3 [5] allows expression of policies in Datalog and uses provenance of facts that are retrieved from another location. Therefore the $_{dg}$FOAF approach would be expressible in SD3. The difference is that $_{dg}$FOAF relies on an RDF representation of data and is restricted to specific policy types, which is crucial for interoperability in the Web. In RT [6], delegation of group administration is allowed similar to $_{dg}$FOAF. However, the RT frameworks lacks for describing groups that have more than one administrator. KAOS [11] uses RDF representations for its policies as $_{dg}$FOAF does. However KAOS assumes a specified policy enforcement point, while in $_{dg}$FOAF this task can be fulfilled by any actor.

10 Conclusion

In this paper, we have presented $_{dg}$FOAF as an approach for a distributed group management based on FOAF. Based on a real world scenario, we have derived the

[4] http://www.dbai.tuwien.ac.at/proj/dlv/
[5] http://www.opensocial.org/

requirements for such a distributed management of groups and introduced the concepts and schema of $_{dg}$FOAF. In addition, we have developed mechanisms for merging policies of distributed groups and determining group membership. We have demonstrated the use and implementation of our approach in the context of an application for access control. With $_{dg}$FOAF, we achieved an important step towards distributed management of groups that is a) independent of concrete platforms and b) supports the usage of the groups over these platforms. In our future work, we plan to further enhance the $_{dg}$FOAF approach and to provide a discovery strategy for finding all relevant $_{dg}$FOAF profiles belonging to one group.

Acknowledgement. This research has been co-funded by the EU in FP7 in the WeKnowIt project (215453).

References

1. Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of ACM CCS '99*, New York, NY, USA, 1999.
2. C. Bizer and R. Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, 2009.
3. D. Brickley and L. Miller. FOAF vocabulary specification 0.97, 2010. http://xmlns.com/foaf/spec/20100101.html.
4. M. Hausenblas, W. Halb, Y. Raimond, and T. Heath. What is the Size of the Semantic Web? In *Proceedings of I-Semantics*, pages 9–16, 2008.
5. T. Jim. SD3: A Trust Management System with Certified Evaluation. In *Security and Privacy, IEEE Symposium on*, Los Alamitos, CA, USA, 2001.
6. N. Li and JC Mitchell. RT: a Role-based Trust-management framework. In *DARPA Information Survivability Conference and Exposition.*, 2003.
7. J. Preece. *Online Communities: Designing Usability, Supporting Sociability.* John Wiley, 2000.
8. D. Recordon and B. Fitzpatrick. Openid authentication 1.1, May 2006. http://openid.net/specs/openid-authentication-1_1.html.
9. H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF+TLS: RESTful Authentication for the Social Web. In *SPOT2009 Workshop at ESWC*, Heraklion, 2009.
10. G. Tummarello, C. Morbidoni, P. Puliti, and F. Piazza. Signing individual fragments of an RDF graph. In *WWW*. ACM New York, NY, USA, 2005.
11. A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. In *POLICY '03 Workshop*, Washington, DC, USA, 2003.
12. S. Wasserman and K. Faust. *Social network analysis.* Cambridge University Press, Cambridge, 1994.
13. C.A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. Decentralization: The Future of Online Social Networking, 2009. W3C Workshop on the Future of Social Networking Position Papers, 15-16 January 2009, Barcelona.